# Performance Results on the Intel Touchstone Gamma Prototype

D. H. Bailey, E. Barszcz, R. A. Fatoohi, H. D. Simon and S. Weeratunga
NASA Ames Research Center
Moffett Field, CA 94035

**Abstract**

This paper describes the Intel Touchstone Gamma Prototype, a distributed memory MIMD parallel computer based on the new Intel i860 floating point processor. With 128 nodes, this system has a theoretical peak performance of over seven GFLOPS. This paper presents some initial performance results on this system, including results for individual node computation, message passing and complete applications using all nodes. The highest rate achieved on a multiprocessor application program is 844 MFLOPS.

## Overview of the Intel Touchstone Gamma System

In spring of 1989 DARPA and Intel Scientific Computers announced the "Touchstone" project. This project calls for the development of a series of prototype machines by Intel Scientific Computers, based on hardware and software technologies being developed by Intel in collaboration with research teams at CalTech, MIT, UC Berkeley, Princeton, and the University of Illinois. The eventual goal of this project is the "Sigma" prototype, a 150 GFLOPS peak parallel supercomputer, with 2048 processing nodes. One of the milestones towards the "Sigma" system is the "Gamma" prototype. At the end of December 1989, the Numerical Aerodynamic Simulation (NAS) Systems Division at NASA Ames Research Center took delivery of one of the first two Intel Touchstone Gamma prototypes, and it became available for testing in January 1990.

The Intel Touchstone Gamma system is based on the new 64 bit i860 microprocessor by Intel [4]. The i860 has over 1 million transistors and runs at 40 MHz (the initial Touchstone systems were delivered with 33 MHz processors, but these have since been upgraded to 40 MHz). The theoretical peak speed is 80 MFLOPS in 32 bit floating point and 60 MFLOPS for 64 bit floating point operations. The i860 features 32 integer address registers, with 32 bits each, and 16 floating point registers with 64 bits each (or 32 floating point registers with 32 bits each). It also features an 8 kilobyte on-chip data cache and a 4 kilobyte instruction cache. There is a 128 bit data path between cache and registers. There is a 64 bit data path between main memory and registers.

The i860 has a number of advanced features to facilitate high execution rates. First of all, a number of important operations, including floating point add, multiply, fetch and store from main memory, are pipelined operations. This means that they are segmented into three stages, and a new operation can be initiated every 25 nanosecond clock period (except for the 64 bit floating multiply instruction, which has two stages, but which can be initiated only every other clock period). Another advanced feature is the fact that multiple instructions can be executed in a single clock period. For example, a memory fetch, a floating add and a floating multiply can all be initiated in a single clock period.

A single node of the Touchstone Gamma system consists of the i860, 8 megabytes (MB) of dynamic random access memory, and hardware for communication to other nodes. For every 16 nodes, there is also a unit service module to facilitate access to the nodes for diagnostic purposes. The Touchstone system at NASA Ames consists of 128 computational nodes. The theoretical peak performance of this system is thus approximately 7.3 GFLOPS on 64 bit data.

The 128 nodes are arranged in a seven dimensional hypercube using the direct connect routing module and the hypercube interconnect technology of the iPSC/2. The point to point aggregate bandwidth of the interconnect system, which is 2.8 MB/sec per channel, is the same as on the iPSC/2. However the latency for the message passing is reduced from about 350 microseconds to about 90 microseconds. This reduction is mainly obtained through the increased speed of the i860 on the Gamma machine, when compared to the Intel 386/387 on the iPSC/2. The improved latency is thus mainly a product of faster execution of the message passing software on the i860.

Attached to the 128 computational nodes of the NASA Ames Touchstone system are ten I/O nodes, each of which can store approximately 700 MB. The total capacity of the I/O system is thus about 7 GB. These I/O nodes operate concurrently for high throughput rates. The complete system is controlled by a system resource module (SRM), which is based on an Intel 80386 processor. This system handles compilation and linking of source programs, as well as loading the executable code into the hypercube nodes and initiating execution.

The software environment of the Touchstone Gamma system is similar to that of the iPSC/2. The SRM runs Unix System V/386 and features the usual networking facilities including support for the Network File System. Also available is remote host software that allows a user to compile and run from a workstation. The individual nodes run a stripped down Unix-like kernel. Fortran-77 and C compilers, as well as an assembler and linker, are provided on the SRM. The system supports Fortran message passing commands for control of multiple processor execution.

## Single Node Performance

The Fortran compiler (produced by Green Hills) that is provided on the initial NASA Ames Touchstone Gamma system is the pre-production release 3.2. Although it has some scalar optimization options, it does not yet take advantage of advanced features of the i860 such as the pipelining of floating point operations and the utilization of multiple functional units. As a result, single node Fortran performance is not outstanding at the present time. However, it is clear that improved performance can be expected in the future as more advanced compilers are made available. This compiler development is stimulated in part by the potential usage of the i860 in high performance workstations and in other systems that are unrelated to the Touchstone development.

| Program | Error | Time | MFLOPS |
|---|---:|---:|---:|
| MXM | 3.43E-15 | 3.106 | 1.35 |
| CFFT2D | 1.26E-13 | 4.029 | 1.24 |
| CHOLSKY | 2.90E-12 | 1.737 | 0.64 |
| BTRIX | 5.53E-13 | 10.889 | 1.48 |
| GMTRY | 8.63E-14 | 138.550 | 0.82 |
| EMIT | 1.48E-16 | 7.909 | 2.86 |
| VPENTA | 1.19E-14 | 0.550 | 1.18 |
| TOTAL | 3.68E-12 | 166.770 | 0.98 |

Table 1: Single Node NAS Kernel Performance Results

Some results of tests using the NAS Kernel Benchmark Program are shown in Table 1. This benchmark assesses the performance of a computer on seven subroutines that are typical of computational fluid dynamics computations done at NASA Ames [2]. The overall single node performance figure of 0.98 MFLOPS (64 bit), which is only about 1.6% of the theoretical peak performance of the i860 on 64 bit data, indicates there is considerable room for improvement in the compiler's effectiveness on this benchmark. These figures were obtained by compiling with no optimization. When compiled with all optimizations enabled (-OLM), the first three figures increased to 5.39 MFLOPS, 3.77 MFLOPS, and 1.71 MFLOPS, respectively, but the remaining tests did not complete, most likely due to a bug in the compiler. By comparison, the overall single node performance figure on the Cray Y-MP for this benchmark is 97 MFLOPS with no tuning and 160 MFLOPS with minor tuning.

Performance results for some simple Fortran loops (see Table 2) on a single node of the Touchstone system are shown in Table 3. For comparison, performance figures are shown on these same loops for several different systems as well as the Touchstone. The Silicon Graphics (SGI) 4D-25 system included in the list employs the MIPS R2000 processor.

Additional insight into single node performance is provided by results for two simplified application programs, which are shown in Table 4. The first program (RELAX) performs a four color cell

| Loop No. | Operation |
|---|---|
| 1 | $a_i = \alpha b_i$ |
| 2 | $a_i = b_i c_i$ |
| 3 | $a_i = \alpha(b_i + c_i)$ |
| 4 | $a_i = b_i(c_i + d_i)$ |
| 5 | $a_i = \alpha b_i + \beta c_i$ |
| 6 | $a_i = \alpha b_i + c_i d_i$ |
| 7 | $a_i = b_i c_i + d_i e_i$ |
| 8 | $a_i = \alpha b_i + \beta c_i + \gamma d_i$ |
| 9 | $a_i = \alpha b_i + \beta c_i + \gamma d_i + \delta e_i$ |

Table 2: A set of basic operations

| Vector length | Touch-stone | SGI 4D-25 | Cray 2 | Cray Y-MP |
|---|---|---|---|---|
| 64 | 3.5 | 0.8 | 86.3 | 187.4 |
| 128 | 3.6 | 0.8 | 91.0 | 187.4 |
| 256 | 3.8 | 0.8 | 97.7 | 197.1 |
| 512 | 3.9 | 0.8 | 112.9 | 199.5 |
| Average | 3.7 | 0.8 | 97.0 | 192.8 |
| Best | 7.7 | 1.4 | 209.9 | 247.8 |
| Worst | 0.9 | 0.5 | 38.3 | 102.5 |

Table 3: Performance (in MFLOPS) of Basic Operations on a Single Processor of Various Machines

| Domain size | Touch-stone | SGI 4D-25 | Cray 2 | Cray Y-MP |
|---|---|---|---|---|
| RELAX | | | | |
| $64 \times 64$ | 6.4 | 2.6 | 109.6 | 177.6 |
| $128 \times 128$ | 6.4 | 2.5 | 112.9 | 190.1 |
| $256 \times 256$ | 5.9 | 2.5 | 115.2 | 190.6 |
| ADI | | | | |
| $64 \times 64$ | 3.4 | 2.0 | 85.6 | 130.8 |
| $128 \times 128$ | 3.1 | 1.9 | 88.9 | 135.2 |
| $256 \times 256$ | 2.2 | 1.4 | 95.2 | 136.5 |

Table 4: Performance of Two Codes on Single Processor of Various Systems

relaxation scheme for the solution of the Cauchy-Riemann equations. The second program (ADI) performs the ADI scheme for the solution of the diffusion equation. Results are listed for comparison on several different single processor systems. All of these results are for 64 bit data.

Some indication of the potential for single node performance is given by the results of the LINPACK benchmark. Using all Fortran, 40 MHz i860 nodes, and no optimization, the performance is 2.6 MFLOPS for 64 bit data. With all three optimization options enabled (-OLM) this figure rises to 4.5 MFLOPS. Corresponding figures for 32 bit data are 2.7 MFLOPS and 5.3 MFLOPS, respectively. When an assembly coded DAXPY routine is employed in the inner loop, 8.8 MFLOPS is obtained on 64 bit data. Somewhat higher performance could be expected by utilizing assembly code for other routines, such as ISAMAX.

The effect of the cache size and the limited bandwidth between the i860 and main memory can be seen by some results of a double precision dot product coded three different ways, as shown in Figure 1. The top two curves are assembly coded where one vector is loaded from cache and the other from main memory. Curve three is also assembly coded but bypasses cache and loads both vectors from main memory. The fourth curve is a Fortran coded dot product compiled with full optimization.

When one vector is loaded from cache (i.e. the top two curves), the dot product peaks at about 27 MFLOPS. As the vector length exceeds the cache size (8 KB = 1000 words), performance drops off dramatically. With a stride of two, only half of the data in cache is usable and so performance drops off when the vector length exceeds 512 words. The Fortran coded dot product also shows the effect of cache and peaks at 8.7 MFLOPS.

Curve three is the assembly coded version that bypasses cache. It remains flat after an initial startup and runs about 13 MFLOPS independent of vector length. In fact, it is the fastest dot product for vectors longer than 1500.

From the above results, it is clear that while the single node Fortran performance of the Touchstone system is much higher than on previous Intel hypercube systems, it still lags far behind the ultimate potential of the i860 system. Certainly there is much work to be done on the Fortran compiler to enable it to effectively utilize the advanced features of the i860. The LINPACK benchmark results, for example, suggest that speedups of a factor of two or three should be possible by fairly straightforward compiler enhancements.

However, even when these compiler enhancements have been implemented, the limited main memory bandwidth and cache structure of the individual nodes will continue to pose a challenge to those wishing to approach peak performance on i860 systems. For example, the 64 bit LINPACK benchmark performance using an assembly-coded DAXPY is still only about 15% of the peak. To obtain significantly higher results, it will be necessary to improve data locality, by better utilizing the registers and cache and minimizing accesses of data in main memory.

It is possible that compiler technology will eventually be sophisticated enough to automatically block calculations so as to maximize data locality and thus boost performance for some codes on systems such as the i860. Indeed, some groups are working on such compiler technology. In the near future, however, the only practical way for users of i860 systems to obtain large fractions of the peak performance will be to restructure computations with algorithms that better preserve data locality. For many applications, it may not be possible to restructure computations in this manner. In such cases only about 10 MFLOPS or so can be expected on a single i860 node.

An example of the potential for higher performance to be had by restructuring a calculation for improved data locality is indicated by some work in progress at NASA Ames by one of the authors (Bailey) and Paul Frederickson of RIACS. They are developing a high-performance fast Fourier transform (FFT) routine for the Touchstone system. In some initial i860 FFT efforts by others, performance rates as high as 40 MFLOPS have been obtained for an unordered (bit reversed), 32 bit, small-sized FFT, but the rate for data sizes larger than the cache drops to only about 10 MFLOPS. The rates for ordered results or for 64 bit data are even lower. Although the NASA Ames work is at present incomplete, preliminary results indicate that by employing an advanced FFT algorithm that preserves data locality [1], performance rates of approximately 40 MFLOPS can be sustained on an ordered 64 bit FFT of any size up to the 8 MB main memory capacity. It is hoped that about three GFLOPS can be obtained by using all 128 processors.

| Length | Distance = 1 | | Distance = 7 | |
|---|---|---|---|---|
| Length (words) | Time (sec) | Rate (MB/s) | Time (sec) | Rate (MB/s) |
| 1 | 9.00E-5 | 0.089 | 3.30E-4 | 0.024 |
| 2 | 9.00E-5 | 0.178 | 3.30E-4 | 0.048 |
| 4 | 9.00E-5 | 0.356 | 3.30E-4 | 0.097 |
| 8 | 1.00E-4 | 0.640 | 3.30E-4 | 0.194 |
| 16 | 2.30E-4 | 0.557 | 4.60E-4 | 0.278 |
| 32 | 2.80E-4 | 0.914 | 4.60E-4 | 0.557 |
| 64 | 3.90E-4 | 1.313 | 5.80E-4 | 0.883 |
| 128 | 5.50E-4 | 1.862 | 7.80E-4 | 1.313 |
| 256 | 9.40E-4 | 2.179 | 1.16E-3 | 1.766 |
| 512 | 1.67E-3 | 2.453 | 1.88E-3 | 2.179 |
| 1024 | 3.13E-3 | 2.617 | 3.33E-3 | 2.460 |
| 2048 | 6.05E-3 | 2.708 | 6.26E-3 | 2.617 |
| 4096 | 1.19E-2 | 2.751 | 1.21E-2 | 2.704 |
| 8192 | 2.36E-2 | 2.776 | 2.38E-2 | 2.750 |
| 16384 | 4.70E-2 | 2.788 | 4.72E-2 | 2.775 |
| 32768 | 9.38E-2 | 2.794 | 9.41E-2 | 2.787 |
| 65536 | 1.88E-1 | 2.797 | 1.88E-1 | 2.794 |
| 131072 | 3.75E-1 | 2.798 | 3.75E-1 | 2.797 |

Table 5: Communication Performance

## Communication System Performance

As mentioned earlier, the routing network of the Touchstone Gamma Prototype is identical to the routing network of the Intel iPSC/2. The main difference between the two machines is the lower latency in the Touchstone system, due to the faster i860 processor.

To measure the communications latency of this system, the time for passing a message between two nodes has been measured for various message lengths. In these experiments, no other communications or computations were performed. The resulting figures are given in Table 5. In this table, the message length is the number of eight byte (64 bit) words, and the rate is listed in megabytes per second.

The times in the table are averages over one hundred repetitions. Figures 2 and 3 show this infor-

| Computer System | Length (bytes) | Latency ($\mu$ sec) | Time/word ($\mu$ sec) |
|---|---|---|---|
| iPSC/2 | $< 100$ | 350 | 1.60 |
| | $> 100$ | 660 | 2.88 |
| Touchstone | $< 100$ | 90 | 1.50 |
| | $> 100$ | 180 | 2.88 |

Table 6: Linear Regression Messing Passing Parameters

mation graphically. These results show that even in a 128 node cube there is very little difference in the actual message passing time between nearest neighbor and maximum distance communication in the hypercube.

Figure 4 shows a close up of the communication times for short messages. Just like on the iPSC/2 message of length less than 100 bytes are sent immediately, whereas for longer messages ($>$ 100 bytes) the node operating system first check for the availability of memory at the receiving end.

Following Bomans and Roose [3] we model the communication time $T_{comm}$ by a least squares fit of the data according to the model

$$T_{comm}(k) \quad = \quad t_{startup} + k * t_{send}$$

where $k$ is the number of 8 byte words, $t_{startup}$ is the latency and $t_{send}$ is the time per word. We obtain the data in Table 6 (the iPSC/2 numbers are from [3]). Thus we are able to confirm a considerably reduced message passing latency, which is mainly obtained through the increased speed of the i860 on the Touchstone Gamma machine, when compared to the Intel 386/387 on the iPSC/2.

The message passing behavior on a real application is considerably more difficult to assess. As an example we present the test results in Table 7, which were obtained when timing two neighboring nodes (distance = 1) exchanging messages. The numbers in this table are displayed graphically in Figure 5. After an initial increase in communication speed the speed drops off again. This can be explained as follows: as soon as the messages reach a certain length, both nodes start receiving the incoming message, even though they have not yet completed sending the outgoing message. Hence, as soon as both nodes starting processing two com-

| Length (8B wds) | Time (sec) | Rate (MB/s) |
|---|---|---|
| 1 | 1.40E-4 | 0.057 |
| 2 | 1.40E-4 | 0.114 |
| 4 | 1.40E-4 | 0.229 |
| 8 | 1.50E-4 | 0.427 |
| 16 | 3.40E-4 | 0.376 |
| 32 | 3.20E-4 | 0.800 |
| 64 | 5.10E-4 | 1.004 |
| 128 | 6.30E-4 | 1.625 |
| 256 | 1.47E-3 | 1.393 |
| 512 | 1.74E-3 | 2.354 |
| 1024 | 3.20E-3 | 2.560 |
| 2048 | 6.14E-3 | 2.668 |
| 4096 | 1.69E-2 | 1.937 |
| 8192 | 4.36E-2 | 1.502 |
| 16384 | 8.93E-2 | 1.468 |
| 32768 | 1.56E-1 | 1.683 |
| 65536 | 3.52E-1 | 1.488 |
| 131072 | 6.67E-1 | 1.572 |

Table 7: Neighboring Node Message Passing Performance

munication requests simultaneously, the communication speed begins to drop off.

## Multiprocessor Application Performance

In this section, we will discuss and analyze the performance of two application programs, which arise from computational fluid dynamics (CFD), that have been ported to the Intel Touchstone gamma prototype at NASA Ames.

The first application is an iterative solution of linear systems arising from the finite difference discretization of a 2-D and 3-D self-adjoint elliptic partial differential equations on regular grids. This problem can be cast into the matrix-vector form $Ax = b$, where the regular sparse, symmetric, positive-definite matrix $A$ takes the place of the differential operator.

The implementation of this problem on the Intel Touchstone consists of decomposing the tensor product computational domain into logically congruent rectangles and mapping these subdomains onto the network of processors. The mappings are chosen so that subdomains sharing common edges are assigned to processors that are directly connected in the hypercube network. This requires, for stripwise decompositions, a binary-reflected Gray code (BRGC) ring and for rectangular decompositions, a BRGC 2-D mesh embedded in a hypercube. Decomposition in this manner results in homogeneous programming of the processors, equidistribution of the load, minimization of the distance traveled by messages and simplified communication patterns for data exchange between processors.

Two essentially different types of data exchanges are required when these concurrent algorithms are implemented via domain decomposition. For five and nine-point second-order finite difference stencils, stripwise decomposition requires two vector exchanges of internal boundary data per processor per iteration and rectangular decomposition requires four such exchanges. In addition to these pairwise exchanges between processors working on adjacent subdomains, global exchanges are required for reduction operations such as inner products and maximums. These global communication operations are implemented via calls to a high level system library provided by Intel [5]. Routines in this library implement the reduction operations using "e-cube" routing algorithm, which requires only $\log_2 p$ concurrent nearest neighbor communication steps, at the end of which all the processors belonging to the active subcube have the required global value.

The Jacobi pre-conditioned conjugate gradient method, multicolor SOR and SSOR preconditioned conjugate gradient method for 2-D self-adjoint elliptic PDE's have been implemented on the 128-processor Intel Touchstone. Some results of these implementations are presented in Figure 6 and Tables 8 and 9. These results demonstrate the performance potential of the Touchstone system under conditions typical of iterative solutions of PDEs in 2-D domains.

One especially curious aspect of this performance data is that in some cases, the parallel efficiency figure (i.e. the ratio of multiprocessor performance to single node performance) is greater than 100%. This is due to the fact that when a problem of fixed size is divided among processors, the memory size in each processor is reduced, resulting in increased cache efficiency.

The 2-D model problem chosen is the Poisson equation:

| Problem | Number of Processors | | | | |
|---|---|---|---|---|---|
| Size | 1 | 4 | 16 | 64 | 128 |
| $512 \times 512$ | 4.5 | 20.8 | 76.3 | 237.8 | 362.4 |
| | | 117% | 107% | 83% | 63% |
| $1024 \times 1024$ | | 17.7 | 80.2 | 299.1 | 524.2 |
| | | 99% | 112% | 105% | 92% |
| $2048 \times 2048$ | | | 69.6 | 320.4 | 628.5 |
| | | | 97% | 112% | 110% |
| $4096 \times 4096$ | | | | 277.8 | 650.8 |
| | | | | 97% | 114% |
| $8192 \times 4096$ | | | | | 554.7 |
| | | | | | 97% |

Table 8: 2-D Red-Black SOR (MFLOPS)

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y)$$

on a unit square with homogeneous Dirichlet boundary conditions, where $f(x,y)$ is chosen so that the exact solution is $u(x,y) = c_x x_2 + c_y y^2$. The initial iterate is $u = 0$ and an absolute convergence tolerance of $10^{-6}$ on the $L^2$-norm of the scaled residual is used. A five point second-order accurate finite difference stencil is used for the discretization of the PDE.

Results for a 2-D model with mixed derivatives are presented in Table 10. The mixed derivative formula is

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \beta \frac{\partial^2 u}{\partial x \partial y} = f(x,y)$$

This problem requires a nine-point stencil for second order accurate finite difference discretization. In addition, performance data is presented in Table 11 for the solution of 3-D Poisson equation using Jacobi pre-conditioned conjugate gradient method.

The efficient implementation of iterative methods on message passing machines requires that the effects of communication delays be minimized. One way of achieving this is to restructure algorithms in such a way so as to overlap communication with computation. This is effected by first updating the u values along the boundaries of the subdomains and then using non-blocking message passing primitives to exchange data between adjacent

| Problem | Number of Processors | | | | |
|---|---|---|---|---|---|
| Size | 1 | 4 | 16 | 64 | 128 |
| $256 \times 256$ | 5.4 | 21.2 | 64.1 | 138.6 | 180.0 |
| | | 97% | 73% | 40% | 63% |
| $512 \times 512$ | | 21.3 | 79.9 | 245.8 | 365.2 |
| | | 98% | 92% | 71% | 52% |
| $1024 \times 1024$ | | | 83.0 | 312.3 | 548.3 |
| | | | 95% | 90% | 79% |
| $2048 \times 2048$ | | | | 329.5 | 652.5 |
| | | | | 95% | 94% |
| $4096 \times 2048$ | | | | | 658.5 |
| | | | | | 94% |

Table 9: 2-D SSOR PCG (MFLOPS)

| Problem | Number of Processors | | | | |
|---|---|---|---|---|---|
| Size | 1 | 4 | 16 | 64 | 128 |
| $256 \times 256$ | 6.7 | 26.3 | 92.3 | 228.9 | 294.1 |
| | | 98% | 86% | 53% | 34% |
| $512 \times 512$ | | 26.7 | 104.4 | 348.8 | 577.0 |
| | | 99% | 97% | 81% | 67% |
| $1024 \times 1024$ | | | 106.0 | 412.0 | 774.8 |
| | | | 98% | 96% | 90% |
| $2048 \times 2048$ | | | | 423.0 | 832.5 |
| | | | | 98% | 97% |
| $2048 \times 4096$ | | | | | 844.0 |
| | | | | | 98% |

Table 10: 2-D Jacobi PCG with Mixed Derivatives (MFLOPS)

| Problem | Number of Processors | | | |
|---|---|---|---|---|
| Size | 1 | 8 | 64 | 128 |
| $48 \times 48 \times 48$ | 5.3 | 37.2 | 186.5 | 252.4 |
| | | 88% | 55% | 37% |
| $96 \times 96 \times 96$ | | 41.0 | 276.6 | 481.2 |
| | | 97% | 82% | 71% |
| $192 \times 192 \times 192$ | | | 316.2 | 588.9 |
| | | | | 633.9 |
| $192 \times 192 \times 384$ | | | | 94% |

Table 11: 3-D Jacobi PCG (MFLOPS)

subdomains before returning to continue with computation in the interior of the subdomains. Computations requiring the internal boundary values received from adjacent subdomains are delayed until all internal nodes are dealt with.

The second multiprocessor application to be discussed is the NASA Ames code ARC2D. This version of ARC2D solves 2-D Euler equations based on the diagonal form of the Beam and Warming implicit approximate factorization algorithm [6] and is capable of treating general 2-D geometries in either time accurate mode or accelerated non-time accurate steady-state mode.

The 2-D Euler equations written in generalized curvilinear coordinates are:

$$\frac{\partial Q}{\partial \tau} + \frac{\partial E}{\partial \xi} + \frac{\partial F}{\partial \eta} = 0$$

where

$$Q = J^{-1} \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ e \end{bmatrix}$$

$$E = J^{-1} \begin{bmatrix} \rho U \\ \rho u U + \xi_x p \\ \rho v U + \xi_y p \\ U(e+p) - \xi_t p \end{bmatrix}$$

$$F = J^{-1} \begin{bmatrix} \rho V \\ \rho u V + \eta_x p \\ \rho v V + \eta_y p \\ V(e+p) - \eta_t p \end{bmatrix}$$

with $U = \xi_t + \xi_x u + \xi_y v$ and $V = \eta_t + \eta_x u + \eta_y v$ as the contravariant velocity components and where $\tau = t$, $\xi = \xi(x, y, t)$, $\eta = \eta(x, y, t)$ is the transformation from cartesian coordinates $(x, y)$ to general curvilinear coordinates $(\xi, \eta)$. This numerically generated coordinate transformation is chosen so as to produce a rectangular domain in computational space with uniform grid spacing of unit length. Here, $\rho$ is the fluid density, $u, v$ are the cartesian components of velocity, $e$ and $p$ are the total energy and pressure, respectively.

Due to less stringent stability bounds and their consequent ability to obtain solutions which require fine grid spacing for numerical resolution, an implicit time integration technique is implemented in ARC2D. Although time differencing can be either first or second order accurate, only the former is required if steady state solutions are of interest. The first order accurate implicit time differencing of the 2-D Euler equations results in

$$Q^{n+1} - Q^n + \Delta t(\frac{\partial E^{n+1}}{\partial \xi} + \frac{\partial F^{n+1}}{\partial \eta}) \quad = \quad 0$$

where the flux vectors $E$ and $F$ are nonlinear functions of $Q^{n+1}$. The nonlinear terms are linearized in time about $Q^n$ using the first two terms of the Taylor series:

$$\begin{aligned} E^{n+1} &= E^n + A^n \Delta Q^n + O(\Delta t^2) \\ F^{n+1} &= F^n + B^n \Delta Q^n + O(\Delta t^2) \end{aligned}$$

where $A = \partial E / \partial Q$ and $B = \partial F / \partial Q$.

Substituting these expressions in the previous equation, we obtain the delta form of the unfactored algorithm:

$$\left[I + \Delta t \frac{\partial A^n}{\partial \xi} + \Delta t \frac{\partial B^n}{\partial \eta}\right] \Delta Q^n \\ = -\Delta t \left[\frac{\partial E^n}{\partial \xi} + \frac{\partial F^n}{\partial \eta}\right]$$

To simplify the solution process, the unfactored Jacobian matrix is replaced by two one-dimensional operators through approximate factorization:

$$\left[I + \Delta t \frac{\partial A^n}{\partial \xi}\right] \left[I + \Delta t \frac{\partial B^n}{\partial \eta}\right] \Delta Q^n \\ = -\Delta t \left[\frac{\partial E^n}{\partial \xi} + \frac{\partial F^n}{\partial \eta}\right]$$

To further improve the computational efficiency of the numerical scheme, the two block implicit operators are diagonalized, based on the eigensytem of the flux Jacobians $A$ and $B$, through the following similarity transformations $\Lambda_\xi = T_\xi^{-1} A T_\xi$ and $\Lambda_\eta = T_\eta^{-1} B T_\eta$ where $T_\xi$ and $T_\eta$ are matrices whose columns are the eigenvectors of $A$ and $B$ respectively. Substituting for $A$ and $B$ in the above, followed by some simplification, we get

$$T_\xi [I + \Delta t \delta_x i \Lambda_\xi] N [I + \Delta t \delta_\eta \Lambda_\eta] T_\eta^{-1} \Delta Q^{n+1} \\ = R^n$$

where $R^n = -\Delta t[\delta_\xi E^n + \delta_\eta F^n]$ and $N = T_\xi^{-1} T_\eta$.

The diagonalization produces scalar tridiagonal or pentadiagonal and block diagonal inversions in place of block tridiagonal or pentadiagonal inversions, without sacrificing the accuracy of the steady-state solution.

In order to overcome the numerical instability due to nonlinear interactions, artificial dissipation terms are added to the implicit scheme. The nonlinear artificial dissipation model chosen is a one in which second and fourth order dissipation are combined with appropriate coefficients to produce a scheme with good shock capturing capabilities [7]. A linearized form of the artificial dissipation model is added to the diagonal factors, which necessitates the use of scalar pentadiagonal solvers. This produces an efficient, stable and convergent form of the implicit factored algorithm for steady state solution of 2-D Euler equations.

The right hand side vector $R^n$ consists of the contributions from the nonlinear flux derivatives in $\xi$ and $\eta$ directions and the artificial dissipation terms, all of which are evaluated using second order accurate central differences. The discretized equations in the interior of the computational domain are supplemented by boundary conditions derived from the characteristic approach, which are applied explicitly.

For the delta form of the implicit factored algorithm, the steady state solution is independent of the time step $\Delta t$. Therefore a spatially varying time step of the form $\Delta t_o[1/(1 + \sqrt{J})]$ is used to accelerate convergence to the steady state. Here $J$ is the determinant of the metric Jacobian.

In summary, the diagonal form of the implicit factored algorithm consists of first forming the right hand side $R^n$ , then performing a block-diagonal inversion involving $T_\xi$, followed by four scalar pentadiagonal inversions for $\xi$ direction sweep. This is followed by another block-diagonal inversion involving $N$, four scalar pentadiagonal inversions for $\eta$ direction sweep and a block-diagonal matrix-vector product involving $T_\eta$, with resultant vector containing the solution update.

The concurrent implementation of the ARC2D algorithm is achieved by stripwise decomposition of the rectangular computational domain in the $\eta$ direction, followed by mapping of the resultant subdomains onto a BRGC ring embedded in the hypercube. As a consequence of this particular subdomain to processor mapping, each of the multiple, independent pentadiagonal system encountered during the $\xi$ direction sweeps is local to the individual processors, and is amenable to solution by Gaussian elimination without pivoting, with no inter-processor communication. The rows of the pentadiagonal systems encountered during the $\eta$ direction sweeps are distributed across the processors embedded in the ring. A data transpose operation, involving heavy inter-processor communication, is performed prior to Gaussian elimination step to gather complete systems onto individual processors.

The resultant solution vectors are scattered across the processor ensemble through a reverse transpose operation, again involving substantial inter-processor communication. The block diagonal inversions and matrix-vector product encountered in the diagonal implicit factored algorithm are computed without incurring any inter-processor communication costs. Computation of the right hand side vectors requires only pairwise exchanges between processors working on adjacent subdomains. Global exchanges are required for computing the 2-norm of the residual of the continuity equation and the number of supersonic points in the flow field, which are used to monitor convergence.

## Conclusion

With the Intel Touchstone Gamma system, multi-GFLOPS peak floating point performance is now available on a MIMD hypercube computer system. Initial performance results indicate that a significant fraction of this peak performance may be obtained on some specialized applications, particularly those that can be implemented with algorithms and techniques that possess a high degree of data locality. For the larger class of applications that do not possess high degrees of data locality, performance rates will be limited by both the restricted bandwidth between the processor and main memory on the individual nodes and by the restricted communication bandwidth between nodes. For both classes of applications, performance rates for the time being are lower than ideal due to an immature Fortran compiler.

On the other hand, such limitations are typical of an early prototype system. Hopefully future developments, both hardware and software, will alleviate some of these bottlenecks and permit broad classes of scientific computations to run at multi-GFLOPS speeds.

**Acknowledgment**

The authors acknowledge the valuable information and assistance provided by Paul Frederickson of RIACS and David Scott of Intel Scientific Computers.

## References

1. Bailey, D. H., "FFTs in External or Hierarchical Memory", *Journal of Supercomputing*, vol. 4 (1990), p. 23 - 35.

2. Bailey, D. H., and Barton, J. T., "The NAS Kernel Benchmark Program", *NASA Technical Memorandum* 86711 (August 1985).

3. Bomans, Luc and Roose, Dirk, "Benchmarking the IPSC/2 Hypercube Multiprocessor", *Concurrency*, vol. 1 (1989), p. 3 - 18.

4. *i860 64-Bit Microprocessor Programmer's Reference Manual*, Intel Corporation, Santa Clara, CA, 1990.

5. *IPSC/2 User's Guide*, Intel Scientific Co., Beaverton, OR, 1989.

6. Pulliam, T. H. and Chaussee, D. S., "A Diagonal Form of an Implicit Approximate Factorization Algorithm", *Journal of Computational Physics*, vol. 39 (1981), p. 347 - 363.

7. Pulliam, T. H., "Efficient Solution Methods for the Navier-Stokes Equations", Lecture Notes for The Von Karman Institute for Fluid Dynamics Lecture Series: Numerical Techniques for Viscous Flow Computation in Turbomachinery Bladings, Jan. 20 - 24, 1986.

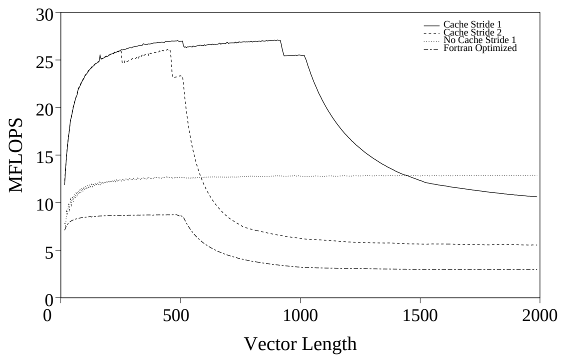

Figure 1: Double Precision Dot Product.

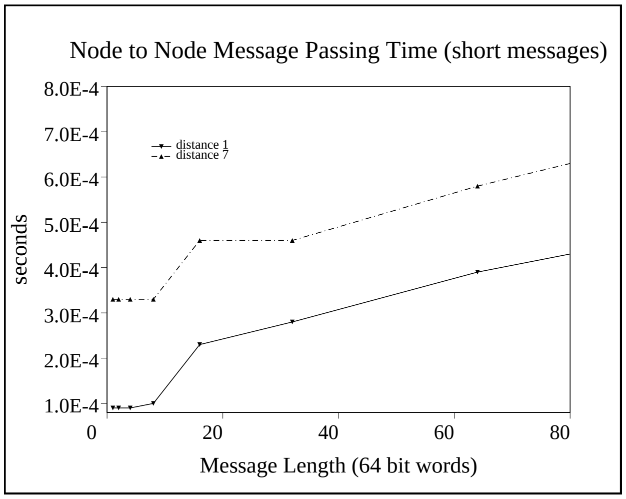

Figure 4: Communication of Short Messages.

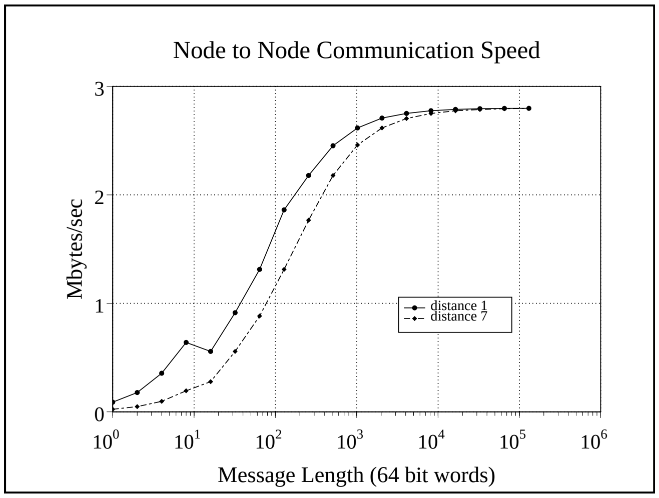

Figure 2: Node to Node Communication.

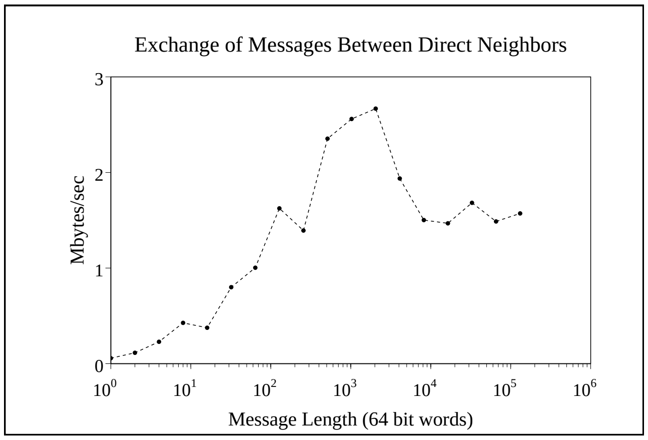

Figure 5: Exchanging Messages between Neighbors.

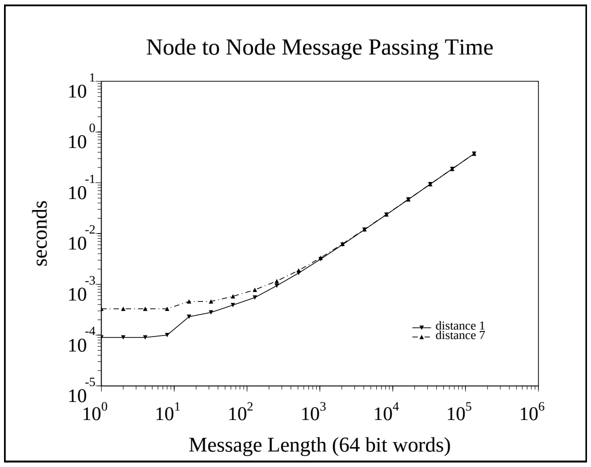

Figure 3: Multiple Hop Communication.